

COMPLEXITY IN MODELLING AND ANALYTIC  
USE OF COMPUTERS

Wm. Orchard-Hays

February 1976

WP-76-9

Working Papers are internal publications intended for circulation within the Institute only. Opinions or views contained herein are solely those of the author.



## COMPLEXITY IN MODELLING AND ANALYTIC USE OF COMPUTERS

Wm. Orchard-Hays

### Foreword

There exists today a wealth of concepts, methods, techniques and tools -- including computerized systems -- which are suitable or even intended for what we now call system analysis. It would seem that virtually any complex planning or evaluation problem could be attacked with some form or other of model and meaningful results produced. This assumes, of course, that reasonably reliable data can be obtained, which in practice is often a severe bottleneck. But even ignoring the question of data, the process of actually formulating, implementing and using a model for the analysis of a complex real-world situation is enormously difficult.

This difficulty is caused by complexity of several kinds. The complexity of the real world cannot be defined away and, indeed, it is the object of system analysis. But there are further levels of complexity which are caused by the concepts, methods and techniques themselves and, more particularly, how they are handled in computerization. To a considerable degree this is tied up with confusion and ambiguity induced by the various representations which are used. Representations are neither concepts nor reality, no matter how narrowly these are construed, and yet, from the first touch of pencil to paper to the reams of printed output from a computerized system, representations are all we actually deal with.

Several computerized systems of enormous power are available to IIASA, often at almost no cost. Indeed, two or three, at least, are here now waiting to be used. Yet no one is using them. Nor is this situation unique to IIASA. This writer has spent a quarter of a century in developing increasingly powerful and flexible systems, and has been assisted by numbers of highly competent people at different times. Similar efforts by

numerous other individuals and groups could be cited. Increasingly, in the last few years, these systems have tended to become monuments to complexity and futility. Clearly something is wrong and this is a matter of deep concern. Certainly we can (or may have to) stop building systems. But such capability is sorely needed in analyzing the enormous problems facing the world.

Only three possible explanations for this situation present themselves:

- (a) There are almost no analysts who are capable of formulating models of sufficient power to utilize big systems.
- (b) Computer technology has become so complicated that most analysts cannot -- or refuse to devote the effort needed to -- really understand it. Hence they are unaware of what is available and what can be done.
- (c) The systems are poorly designed with respect to the kind of work analysts must undertake.

We must believe that (a) is false or else we may as well all pack up. On the other hand, it is increasingly clear that there is some truth in both (b) and (c). The following discussion is aimed at clarifying both sides.

Those who are bored with discussions bordering on the philosophical may wish to only skim over sections 1 and 2. Section 3 is similar in vein but short and it provides a necessary preface to the sequel.

## I. The Puzzle of Complexity

Complexity is characteristic of our time. This phenomenon is not confined to any one sphere of activity, any one area of intellectual pursuit, or any one cultural, political, or technological environment. It is, of course, more predominant in some areas -- both physical and conceptual -- than in others, but, nevertheless, complexity is an ever-encroaching cancer on human experience in the twentieth century, and particularly since WWII.

Complexity, in the sense meant here, is not at all synonymous with difficulty, sheer size, or extensive administrative details. A few people have always been able to surmount incredible obstacles

and, in this sense, to solve difficult problems. History is full of such stories in various fields. Massive enterprises and far-flung but efficient administrative organizations have been known since antiquity. But until recent modern times, simplicity and uniformity were much more the results of achievement than complexity.

In science, the early results in almost every field, beginning with physics in the sixteenth century, were marked by simplicity, elegance, and apparent generality. In technology, as late as the 1930s, it was claimed that all mechanisms depended on a small set of basic devices or principles (some number in the teens as I recall). In mathematics, some complexity is inherent but, initially, this was more in individual problems which were difficult or unfamiliar rather than in a confusing maze. (Of course, there were holdovers of confusion from antiquity.)

As a succession of brilliant minds developed more and more general mathematical methods -- largely motivated by problems in the physical sciences or practical problems occurring in the conduct of human affairs -- these were seen both as unifying concepts and as a confirmation of Galileo's contention that the "Book of Nature is written in mathematical characters". The idea emerged that mathematical statements represent a model of reality and not merely a method of solving practical or theoretical problems. (This is over and above the concepts and techniques of "pure" mathematics or its foundations in logic.) This conviction is still very much a basis of current work even though the increased use of mathematics in the less exact sciences, such as economics, requires one to hedge somewhat about the validity of a model. However, this is not viewed as a weakness in the concept of mathematical representations but as a difficulty in formulating a model of a situation for which experimental methods are impractical and precise laws unknown. Hence one must rely on historical or indirect data and opinions, both to formulate and validate theory.

It is true that conceptual difficulties have arisen in physics, mathematics and logic which have, at least for a time, had a disquieting or even shattering effect on scientists -- so-

called crises. But the idea that correct mathematical representations are inherently valid and consistent -- essentially as an article of faith -- has been strong enough to survive all such shocks. Indeed the practical applications have hardly been disturbed at all. (Many scientists and mathematicians might deny that science is grounded in essentially intuitive beliefs. Yet to some of us at least, this conclusion seems inescapable when observing the methods of science. This is not raised as an objection in any sense.)

Nevertheless, new crises are forcing themselves on us, due at least in part to past successes. First, the enormous growth in human knowledge has, in itself, forced specialization, in some cases extreme. It is not possible for a twentieth-century Leonardo to emerge, or even a Gauss. No one person, even of the greatest genius, can comprehend a broad spectrum of fields in sufficient depth to make fundamental contributions to them all. This increases the complexity of communication and cross-fertilization of ideas. It is difficult even to know whether an applicable theory or method for a problem at hand has already been developed. Scientific competition also contributes to the difficulty.

Second, the growth of industrial technology, urban culture, population, and other factors often noted, has created new kinds of problems. These problems do not respond to the kinds of models in classical physics, for example. Such models not only elucidated but anticipated facts. Perhaps the first dramatic case of this kind was the "discovery" of Neptune. A more precise case is the bending of light around a large body as predicted by relativity theory. For pure imaginative abstraction, it is hard to outdo Dirac's famous equation from which results jump out like "rabbits from a magician's hat", as it is described in prestigious works. Thus "facts" may be based on perusal of mathematical formulae rather than on observations. Of course this is one purpose of a model, perhaps the main one. It depends, however, on knowledge of fairly precise laws. When models are applied to areas where such laws are lacking and where various uncertainties must be

taken into account, the situation is quite different. Both observed facts and deduced facts may have low confidence levels with respect to either explanatory or predictive value. This greatly complicates analysis, obviously.

Another source of complexity is related to the development of computers and data processing technology. It is not the complexity of the computers themselves to which we refer, but the perplexing ambiguity in representations which they engender. Since this is the central theme of the sequel, no attempt will be made here to illustrate it in a few words.

Complexity is itself forcing a new crisis upon us. The difficulty is essentially this: The world is faced with enormous problems which, if not soon resolved, threaten the very continuation of human life. All the complicated techniques of analysis and decision sciences, which are now nominally available, are needed to find solutions. The very complexity of these methods and tools, however, inhibit their effective use, and they are difficult to comprehend. At the same time, their potential power seems to be precisely what is needed to resolve the problems facing mankind. This is the puzzle of complexity.

## 2. On the Complexity of Human Interaction with Computers

In the past, complexity has often been dispelled by a fresh approach, a new viewpoint or the recognition of a basic principle. This is what is usually hoped for but, increasingly, it does not work. This is almost characteristic of computer technology. (In hardware technology, significant exceptions to the above statement could be cited: e.g., the transistor and printed circuits.) For well over a decade, in some cases two, the computer has been performing tasks routinely which could not have been done otherwise. But the true potential of the computer has not been even approximately achieved except in a few special cases at enormous effort and cost. (The U.S. space effort is perhaps the best illustration of this.) A long series of terms and concepts have been put forth and many implemented -- integrated data bases, management information systems, artificial intelligence,

etc. -- but almost without exception they have fallen short of expectations. This is not to say that computers are not worth their cost -- certainly they are justified for many purposes. But they tend to constitute a separate technology of their own. They have been most successful in repetitive data processing applications and sheer computing tasks. While a considerable degree of flexibility has been achieved from the computer specialist's viewpoint -- an incredible amount of those of us who worked on the earliest machines -- this has not extended in sufficient measure to an analyst-user.

As early as 1957 or 1958, the term "automatic programming" was coined. It was hoped that the role of the programmer could be virtually eliminated. Today programmers constitute one of the larger labor classes in the U.S. (Most of them are using the "automatic programming" techniques.) A series of languages have been developed to "make the computer more accessible to the user". The most widely-used one is almost the oldest and certainly the least adequate -- FORTRAN. Even IBM who fathered it has tried to disavow it -- without success. This is unquestionably due to a reaction against complexity by computer users. The U.S. government insisted on the development of COBOL for all computers it purchased, to simplify and standardize programming and documentation. Today there are entire floors of large buildings full of COBOL programmers.

Examples could be cited ad nauseum. One case which is very germane to system analysis is linear programming which has been under intense development for almost 25 years. Projects are being started right now to "make use of LP and mathematical programming techniques more accessible to the user". This writer only last year completed (almost) the latest version of a long series of "gee-whiz" systems, this one for interactive use. Yet almost no one is using it. It is now contended that the data management approach, which was considered a significant improvement itself, is at the wrong level of analysis. This may be true but the level now proposed will either have to build on or essentially duplicate the complex system already in existence.



This is typical of application systems and perhaps even more so of basic software.

Certainly many talented people have devoted their efforts to improving software, and computer science has already made significant theoretical contributions in a number of areas. An enormous body of literature exists on computers, computer based applications, and algorithms of various kinds. Although of mixed quality, much of it is in good scientific tradition. But all this seems to help very little in data management.

Actually the physical capability for storing and processing enormous amounts of information (or at least data) now exists and is in use. One difficulty is that computerized data, at least if it is to constitute information or be used to calculate meaningful results, is in effect procreative -- and very prolific. There is more data about data than about reality. Since the use of data also involves concepts of some kind, the concepts themselves must have representations and these constitute more data. Furthermore, the more comprehensive and powerful the concepts, the more information is required to utilize them. A simple and familiar example is a matrix. One can conceptualize problems in terms of matrix algebra with relatively few symbols. But if the concepts are to have practical application, actual arrays of numbers must be provided and processed. These numbers come from somewhere and must be identified. The results produced must be related in some meaningful way to the problem, which means either words or charts that people can read.

One major reason for the resulting complexity is that almost no simplifying or unifying concepts exist for the handling of data itself. This writer chaired a committee that worked on this problem for many months in the late 1950s. A recent perusal of current literature on the subject revealed that almost no real progress has been made since. If anything, it is more confused than ever since a number of specialized terms have been introduced which are only labels -- nothing follows from them. They are much like Euclid's definitions but far less intuitive.

Computing and data processing, of course, cannot be blamed for the complexity of modern life. It could even be argued

convincingly that, quite the contrary of being a cause of complexity, this technology has arisen in response to it as a means of handling the enormous computational, record-keeping and information requirements of the modern world. Certainly there is no intent here of faulting the computing industry and profession which, in one generation, have achieved more than probably any other field in the history of mankind in a similar time span. This foreshortening of traditional growth periods is itself a major cause of complexity since there is simply not time to sort everything out in an orderly fashion. However, this is not unique to the computing field. The automotive, aeronautical and radio-television technologies, for example, developed in comparable time spans and introduced qualitatively different aspects into human life. But none of these require the user or consumer to interact with the technology itself in an intimate or complicated way -- regardless of how the technology may have altered life-styles. This is true also of the commercial applications of data processing. The case with scientific or analytical use of computers, however, is different. It is this area to which we will now confine our attention.

### 3. Concepts vs Reality

If an intelligent but philosophically unsophisticated (uncomplicated) person were asked the difference between concepts and reality, he would probably feel that a clear distinction could be made. On some further reflection, he might concede that the distinction is not, after all, absolutely clear. If I look out the window at the landscape and the town, I feel I am viewing reality. If now I turn back to the mathematical or programming problem on my desk, I have an equally strong feeling of reality about the abstract domain under contemplation. This is a familiar experience to those of us who spend our time at mental rather than physical tasks -- and we are now in the majority in many areas of the world. While we certainly feel a difference in the experience of studying and that of walking in the woods, both are somehow real. In fact, since we probably spend more time at a desk than in the outdoors, the latter may seem less

real to us than the former. A common feeling two or three days after returning from an exciting trip is that it was just a dream.

The world of the computer has added new dimensions to this ambiguity. Suppose a scientist needs to make certain calculations using some well-known mathematical method -- a set of formulae or an algorithm, say. It is perfectly clear to him that the situation he is modelling (whether he calls it that or not) is only abstractly and imperfectly represented by his model -- that is, the model is a concept representing certain aspects of reality. It is also clear that the method is a concept but a more "real" one since its validity depends on well-established proofs and not on any particular application or interpretation. He also knows that the method can be (or has been) programmed and that the computer can calculate numerical results for a number of cases which he wants to examine. The whole set of ideas (another concept) may occur to him in a flash, without any conscious separation into the above steps.

In order for this conceptual plan to become "reality", the investigator must first reduce a number of things to writing. (This use of "reduce" always seems inappropriate. "Expand" is closer to what happens.) The exact order will depend on style but let us suppose he first writes down the general formulae. It is most likely that he will do this in standard mathematical notation which, however familiar, is highly abstract and condensed. Seeing that the method is indeed appropriate to his conceptual model (of course, this whole scenario is grossly oversimplified), he next writes "where:", followed by a string of argument, set, and parameter definitions. It is at this point that implementation difficulties begin to appear.

#### 4. Representations

For the present, let us assume that our user's method has indeed been programmed for the computer and that the mathematical formulae are effectively built into the routines, and hence require no explicit specification. Later, we must consider the

more important case where this assumption is not completely true.

With the above assumption, then, our user need not do any actual computer programming. However, the full assumption only makes sense in the event that the method is quite general and the expense of developing a general application system ("package") could be justified on the basis of a large number of expected users. Thus we are not considering the case where standard sub-routines for trig functions, or Bessel functions, or something similar, are available. In such a case the user would still have to do computer programming. The assumption implies that, at the most, the user might have to write a simple and stereotyped control program to define array sizes and source files.

Now it is clear that any method will require input data and must produce some form of output. Let us further assume that output is standardized and requires at most some simple input parameter to specify frequency of output or perhaps one or more of several predefined formats. Then we can concentrate on input data.

The first question our user encounters is what "language" he should use in specifying sets, arguments, parameters and, perhaps, source data. This may be further complicated if "identifiers" for variables are required, which may be necessary to identify output, for example, or simply due to system conventions. In any event, some sort of translation and transcription from the user's natural mode of definition to the system's conventions will be required. The designer of the system, no matter how competent and familiar with the application area, had to establish conventions; these were probably adapted from the notation of some leading authority in the field, modified for limitations inherent in machine-readable character sets.

If one writes

$$\sum_{j=1}^n a_{ij}x_j \leq b_i \quad , \quad i \in \{1, \dots, m\} \quad , \quad (1)$$

anyone with modest mathematical training will understand what it signifies, assuming the context has been made clear. This writer,

on seeing the above notation, would assume it had something to do with a linear programming model, which might not be so. But assume it is. With only minor variations, it could represent the constraints in any LP model. It is nothing but the classical statement of LP constraints according to one school of writers. As a representation, the most it represents is a part of a methodological concept, an abstract notation of an abstract idea. Outside of a mathematical text or statement of method, it represents nothing at all, even if transliterated to "computerese".

To further clarify this important point, consider actual numerical data. We will ignore problems of format; it is sufficient, for example, to assume that everyone knows and accepts FORTRAN conventions. Suppose one has an array of numbers, for example:

2.13	1.00	-1.25	
-0.15	0.85	1.01	(2)

What do they mean? We can "read" them, of course, and so, in essence, can the computer. Before that is really possible, though, a convention must be established as to whether they are presented by row or column, and what the index limits are, since the computer really gets them in a linear string. Assuming all this has been conventionalized and specified, the above array is still just six numbers. We feel that there is a "reality" to actual numbers but, in fact, (2) is no more meaningful than (1). The array (2) could be an instance of any 2 x 3 table or matrix.

Thus the meaninglessness of representations is inherent whether it occurs for very general, abstract concepts, or very specific, "real" values. There is a gulf between a representation and what it signifies which cannot be bridged in a mechanical or automatic way. The nearest approach to such a connection is a widely accepted convention and perhaps a procedure. Thus, (1) is given meaning by mathematical training and made "real" by a system of computational routines; (2) is given structural

meaning by a set of conventions and input routines, and given specific meaning by what a user construes the values to mean.

## 5. Some Definitions

Even the above more-or-less obvious comments do not indicate the full degree of ambiguity in representations or the complexity it creates. In order to discuss this further in a meaningful way, a few precise definitions are required. They are, to some extent, arbitrary but, since universally accepted definitions do not exist, hopefully no harm is done by using common words as labels for specific meanings included in their general senses.

Concept A mathematical or mathematical-like mental formulation of considerable generality but specific enough to be articulated in a way readily understood by a knowledgeable group of people.  
Example: "We can treat this class of models as mathematical programming problems with a quadratic functional and linear constraints."

Theory A carefully formulated and formally proved set of mathematical or logical concepts which can be applied to any problem which meets or can be construed to meet the stated formulation, assumptions and conditions. Essentially a set of theorems and hypotheses. Examples: "The Theory of Groups", "Standard Statistical Methods,", "Integer Programming"; also applied to the specific theory -- possibly hybrid -- on which a method or set of related methods is based. Not to be construed so broadly as, for example, the Theory of Electrodynamics.

Method A procedure widely known, at least in its basic form, for a particular type of problem, usually one which is commonly programmed for computers. Examples: "The method of Least Squares", "The Simplex Method", also applied to concepts of computer science, e.g., "the method of inverted

files," "compilation methods".

Algorithm A method which is either inherently iterative and terminates by convergence, such as one employing some variant of Newton's method, or which is commonly implemented by a procedure involving (even theoretically) a finite or fixed number of steps which, however, cannot be prespecified in detail.

Application System An elaborate and coordinated set of computer programs which carry out a method or a set of related methods of a theory, plus providing some degree of data management, control, and report writing capability, sometimes extensive.

Package Similar to an application system but more loosely connected. Essentially a related set of individual programs which may require some additional programming in order to utilize them in a particular situation.

Interactive System Either a basic hardware/software system designed for interactive use from terminals (then better called an interactive environment), or an application system implemented in such an environment. In contradistinction to batch processing or applications implemented for batch processing.

(Mathematical) Model Here restricted to mean a specific application of a theory for which a computer-implemented method or related set of methods exists, or can be created using known technology. See further in next section.

Reality That part of a real-world situation which is abstracted for study using a model.

Further definitions must be postponed until certain notions already introduced are clarified.

## 6. Regarding Mathematical Models

The term model, even when restricted to mathematical models, is commonly applied in different senses which, though related,

involve different viewpoints and techniques. One cause of confusion and complexity is that these different senses cannot be put into either a strict hierarchical classification or a fully ordered time sequence. This has almost nothing to do with the nature of the reality under study but is characteristic of the modelling techniques themselves. Since the purpose of a model is almost always to study complicated reality, it is essential to bring as much order and clarity as possible into the modelling methodology. Otherwise, complexity is compounded and the resulting confusion tends to nullify the effectiveness of the whole effort.

Even though modelling methodology cannot be put into a neat hierarchical tree or time ordered sequence, some broad aspects can be quickly recognized and certain precedence relations are obvious. We will begin by cutting away those parts which have only peripheral relevance to the present discussion.

First, an important and comprehensive project of system analysis would very likely require more than one form of model, even though one might be central to the overall approach. We will assume that distinctly different types of models can be and have been segregated in the initial project planning. This is not to deny that overall integration into a system of models may be desirable or necessary at a later stage. However, the ease and effectiveness of this will depend in large part on the quality and operability of the separate models, and how well they can be interfaced. This last consideration is best served by standardizing implementation techniques and data conventions as much as possible, rather than by attempting to combine two disparate methods or theories from the outset. Clearly, mature judgement will have to be applied to this matter in individual cases but, at least for the present, it is assumed that the modelling scheme is manageable in a practical sense within a fairly well defined and proven theory, which may have to be extended somewhat in the modelling process itself.

In connection with the preceding paragraph, those who must explain the methodology used to the client or other important



outsiders, sometimes refer to the entire project as "use of a mathematical model". This gross oversimplification is apparently deemed necessary to remain meaningful in "high-level discussions". Whatever virtue it may have for presentation purposes, this grand view is only a hindrance in trying to analyse complexity and the difficulties of applying modelling techniques. We will have no occasion in the sequel to construe a model in so gross a sense.

Second, those concerned with developing theory, methods, and even application systems, tend to speak of the model. By this they really mean the theory with its abstract formulation and mathematical notation. This viewpoint is, of course, highly useful in conceptualizing but only of value as reference material in an actual application.

Thirdly, a model is regarded as an abstract, i.e. symbolic, formulation of reality (already an abstraction). At this point, we must regard the use of the word as legitimate, even though this form of a model is a far step from actual realization of results. This process of formulation has value in itself in clarifying the scope, resolution (resolving power) and relevance of the approach; in determining what classes of data are required; and in indicating the range of cases or parameter studies which the overall goals demand (e.g., for options and confidence levels). Hence, this abstract formulation is in a real sense a model.

The confusion begins in moving to the next stage. Formulation is a task involving discussion, study, and paper and pencil -- in short, strictly human functions. The symbology thus evolved could, it is true, be transcribed into machine readable form and treated as an abstract definition of the model for the computer, i.e. the application system. This has not been the approach in past technology but is now being considered. Before attempting to assess whether this has any meaning, let us complete the list of requirements for the model as a whole.

The next stage may be termed implementation of the model. At the moment, we are concerned with data but the same term will also be used in a similar sense with regard to method, meaning creation of usable computer routines or an application system,

when this is called for. (There are, in fact, several aspects to the implementation of a model and it is here that confusion can easily arise.) The following tasks must be performed in one way or another, just for data, assuming the method has been implemented.

(1) Source data must be collected or located and identified. This may require auxiliary projects of several possible kinds. We will assume here that basic source data has been brought to the stage of computer readable format with both numerical or symbolic values and all required identifying labels, indices or whatever. There is no intent to minimize the possible difficulty of this part of the project -- which may in fact be the hardest -- but data collection is a separate methodology in its own right. If the pertinent data already exists in an accessible data bank, then, of course, data processing techniques can be applied to obtain it. It should be pointed out, however, that in many cases, at least some data must be available before the process of formulation can be completed. Thus there may be an iterative nature even to data collection.

(2) Source data must be checked and validated in most cases. When "clean" source files are at hand, it will probably be necessary to further process them into forms suitable for the main model. This may require reformatting, aggregating, various kinds of computing, or even implementing preliminary models. Only after all this is done can it be claimed that data for the model is available. Note that the final form depends on existing or planned conventions for input formats to the application system to be used. Hence there may be another time dependency between data preparation and method implementation.

(3) The data must be "input to the model", possibly specialized by case. Here we encounter a confusion not merely of terminology but of concept. What is the "model"?

The question above brings us to the heart of the subject under discussion. Actually, there are two parts to it. We assumed that the method had been implemented, or would be as a project task. In fact, both data and method may involve several

cases so that there is a control problem as well. This requires human interaction and hence an interactive system seems most suitable for modelling activities. (An exception would occur when the main computations involve a series of long runs with clearly specified parameters. However, this would seem to be less and less likely as the realities modelled become more complex, especially with the tremendous speed of current large computers.)

## 7. What is a Model?

After eliminating peripheral senses and assuming away the difficulties of data collection and preparation, we are still left with considerable ambiguity as to what constitutes a model. We will illustrate this with a series of questions which, admittedly, are straw men.

- (a) If the initial formulation of the reality under study is the model, then what is its representation, where does it exist, and how shall we regard the data prepared for it?
- (b) If the methods specialized to the formulation constitute the model, where do these definitions reside, and how are they given meaning? Or must each model have its own application system or package?
- (c) If a model can be considered implemented only after usable data is available, then is its representation and structure a part of the model; if the data exists in separate files, how shall we regard other uses of the same data?
- (d) If the model can only be regarded as a dynamic entity requiring human interaction and monitoring, then is it anything more than a set of machinery which humans operate? If so, then is each instance of use a separate model?
- (e) If the modelling scheme is itself a kind of grand iterative process, then does the model ever have a distinct existence?
- (f) If any of the viewpoints implied by (a) to (e) are adopted, does it make any sense to talk about the reusability or transferability of the model?

Even though these questions are loaded, they do bring out the necessity for clarity in ideas and unambiguous definitions.

To this end, we begin with the following.

Conceptual Model The initial written formulation which defines the abstract reality to be studied, the form of model to be used (in terms of theory and method), and the notation which will be used to relate the components of reality to the theory and method. This conceptual model is then the primary reference document with respect to implementation of method. The notation used (after preliminary explanations) should be suitable for computerized referents. However, the conceptual model is strictly a product of human analysis and intended for human use. Any computerization would only be in the nature of a librarying service.

Source Directory A written document which defines the source, nature, and preprocessing if any, of the data implied by the conceptual model, together with the notation which refers to the final form. This notation must be either identical to or a consistent expansion of that used in the conceptual model. Otherwise, a source directory can have the most varied forms, as required, and may even imply, specify, or reference auxiliary or related projects which furnish data. The source directory is then the primary reference document with respect to implementation of (or possibly merely accessing) data files. It is not itself computerized (except perhaps in the sense of librarying) but defines most of the notation for data in actual computerization.

Model Vehicle The computer system to be used (or the relevant part), including hardware, basic software, and, when appropriate, an existing application system. (Conceivably more than one application system might be used. Usually, however, this will lead to interfacing problems.) If an application system must be created or extended, this is also part of the vehicle. However, special files, control programs, etc. created for the project using an existing system are not part of the vehicle. Network facilities might be included.

Structural Model A second conceptualization, completely divorced from reality, and referring explicitly to computerization. It is defined by a written document, which in turn prescribes the vehicle, and in which symbolic charts, diagrams and tables are used freely. The methods and notations defined by the conceptual model, the data to be made available as described in the source directory, and the known or defined characteristics of the vehicle, are taken to be reality. The (definition of the) structural model is essentially the overall design of the operational model (defined below). In the event that new methods must be implemented, this should be spelled out here too, or else reference made to detailed specifications for the necessary programs.

Note: Software has long existed for computer-produced charts and explanatory text such as might be used in defining the structural model. However, this is only a specialized form of librarying and not actual computerization. Similar but much more elaborate software for such uses as architectural and engineering design also exists and, in this context, is actual computerization. We will assume that the structural model document is written or drawn by people but it is quite possible that output from standard software may be included in the final version, if only for illustration.

-----

Operational Model The fleshed-out, computerized realization of the schema implied by the structural model, checked out and ready for use, and supported by detailed user documentation.

It is clear that there is a large gap between the definitions of structural and operational models, but at least we now know where to concentrate our attention. Also, we need not ask any more silly questions about what a model is. However the operational model is used, it presumably produces results meaningful to the original formulator of the conceptual model. Whether or not "the model" is reusable or transferrable is a moot question.

(This writer's opinion is that usually it is not but substantial parts may be).

Note that all the above definitions are perfectly general and could apply to any modelling project for which computer methods are to be used. In effect, they constitute part of a general management plan. The gap is filled by construction of the operational model and here it is much harder to be general. We discuss this in the next section.

#### 8. Construction of the Operational Model.

Assuming the expected competence in formulating the conceptual model, in defining the source directory and carrying out its tasks, and in obtaining necessary results from the operational model and making valid judgements about them, then the construction phase is the keystone to the whole modelling process. This actually begins with definition of the structural model but this should not be done in too great detail. One needs to get a comprehensible overview of just how the whole operational model will work in principle, what facilities will be required, and what time and cost factors can be expected. This clearly requires the effort of a system analyst with close coordination with the formulator and the data specialists. The role of the system analyst here is comparable to that of an architect working with the client. Once the outline is drawn, however, the detailed design should be left to the expertise of the system analyst.

Now it is just here that the evolution of general-purpose application systems has run into difficulty. The designer of such a system does not have a client, or at best he has a very few at the time, but is trying to design for any of a large number of hoped-for users. A method of considerable difficulty creates plenty of design problems with respect to computational organization and efficiency, handling of a number of possibly large files, staying within the physical or administratively-dictated limits of the operating system, etc. etc. It is enough to handle all this within the extent of one theory and its method or methods. Hence these systems are inherently somewhat specialized.

Consequently, it is not surprising that systems for different theories, or even different systems for the same general area, are not alike and largely incompatible.

Furthermore, it is only after considerable experience with a system -- after other users with different viewpoints have tried to use it -- that flaws in the design begin to be manifest. This is especially true with respect to the degree of human interaction desired and the depth to which this must go. In the early stages of evolution of a type of system, users are glad to get anything that does a complicated job with reasonable efficiency. Once people are accustomed to a capability, however, they then begin to think more generally and may need additional flexibility which is completely incompatible with the original design and of which the designer was never aware (in fact, neither was the early user).

The very considerable capabilities available today would not exist if designers had not made arbitrary decisions. (This holds also for basic software produced by the manufacturers but it doesn't seem to bother people so much any more. "That's just the way computers work." However, individual attempts at replacing basic software are not too successful either and create even greater problems of noncompatibility.) Moreover, the operational control mechanisms that have evolved or been superimposed on systems are actually quite usable today and not a major cause of complaint. The case with data definitions and management, and with algorithm implementation structures, is different. Arbitrary representations and implied meanings have caused confusion and consternation to many users.

Nothing much can be done about all this with respect to existing systems. One must simply evaluate the advantages and disadvantages of using them. If they provide highly-developed and thoroughly proven computational subsystems, this advantage cannot be lightly ignored. The cost of bringing an important application system to such a state is incredibly high. (The money spent on current LP systems, for example, has run into many millions of dollars.) Nor are they easily separated, modified, or extended. (Work is underway now on an LP system which

it is hoped will have this sort of flexibility, but it must still be regarded as experimental.) Packages exist for a number of methods but these provide mostly the basic computational subroutines. Other systems display characteristics similar to the LP systems.

Another important consideration to be taken into account in the detailed design of the operational model is efficiency. In the past, many people have pooch-pooched the question of efficiency, claiming that people-time is more valuable than machine time. This is simply not true. Again, LP provides a good illustration. After the first few years, a fairly reasonable scheme of referents evolved and was generally adopted (LP/90, ca 1960) and then expanded in MPS/360 (ca 1964) which is the basic standard today. However, this form of input is very tedious and is not generalizable. Various data management schemes have been added, none of which have been fully satisfactory or generally accepted. Some of these are quite general in nature but, almost without exception, they are complicated. Efficiency varies; considerable inefficiency can, in fact, be tolerated to gain generality but there are fairly small factors beyond which people are unwilling to pay the cost in processing time. A factor of 4 is probably the maximum, no matter how good the language is. A general language can easily require ten times as much processing time as a simple, stereotyped, linear input stream. Hence, system designers have become wary of generalizations, at least if their product must sell.

Consequently, the detailed design and construction of the operational model for an elaborate modelling project requires, in itself, exercise of expert judgement and making of difficult decisions. If at all feasible and reasonable, cost should not be allowed to be the dominant factor here. The effort and money expended on the other parts of the project are substantial. An operational model which inhibits full investigation of the reality under study, due to inflexibility, unreliability or inefficiency, is much more costly in the long run than additional effort in design and construction.



These things are much easier to say than to do. There is evidently no substitute for an experienced team who have worked together over a considerable period and have developed their own jargon, techniques and system componentry. To the extent that this is incompatible with IIASA's structure and goals, serious thought must be given to what constitutes a meaningful substitute. No amount of theory, documentation, or external collaboration can quite take the place of experienced teamwork.

The benefits to be realized from truly effective, flexible and easily controlled operational models would surely merit a substantial effort to achieve them. The important complexities of reality could then be studied effectively which is what system analysis is all about.

- - -

## APPENDICES

The following pages starting with 1-3 are reproduced from parts of the following document:

SESAME MATHEMATICAL PROGRAMMING SYSTEM  
DATAMAT REFERENCE MANUAL (Third Edition)

Computer Research Center for Economics and Management Science  
National Bureau of Economic Research, Inc. D0087 July 1975.  
(This document is copyrighted 1975 by NBER.)

This writer developed both the SESAME system, with collaboration by William D. Northup and Michael J. Harrison, and its DATAMAT extension, at NBER. The main SESAME system is fully operational and thoroughly tested. The DATAMAT extension is still experimental and not quite complete (particularly the report-writing facility). The system is available to IIASA at the CNUCE center in Pisa, Italy, and we have an account there which can be accessed via remote terminal.

Pages 1-3 to 1-18 are taken from Part I of the document, prepared by Robert Fourer who has been in charge of documentation and testing. It shows how two LP models are handled with DATAMAT.

The "Appendix Overview", pages A-1 to A-26 (out of 34) was written by the present writer towards the end of 1974. It gives additional viewpoints on the problem of representations and discusses most of the "verbs" available in DATAMAT. The report-writing verbs (not complete) and several utility verbs are not included in the part reproduced.

The discussions in these pages further amplify the complexity of practical applications of modelling techniques. Although DATAMAT is certainly not the only approach (or even the most common one) to data management, it is the outgrowth of several development efforts which started as early as 1958-60 and is a direct descendent of elaborate systems developed from about 1965-6. Much of its design was dictated by a large commercial user who studied the problems in depth over a period of some

years. Hence it warrants more than a cursory appraisal.

## EXAMPLE 1. A SIMPLE ASSIGNMENT PROBLEM

Ten women each rate ten men on a scale from 0 to 10. They are to be paired into ten couples in the best possible way - that is, to maximize the sum of the ratings associated with the pairings. More formally, let  $a_{ij}$  be the rating given by woman  $i$  to man  $j$ , and let variable  $x_{ij}$  be 1 if  $i$  is paired with  $j$ , and zero otherwise. An optimal pairing is then determined by the following linear program:

$$\begin{aligned} &\text{maximize } \sum_i \sum_j a_{ij} x_{ij} \\ &\text{subject to } \sum_j x_{ij} = 1 \quad j = 1, \dots, 10 \\ &\quad \quad \quad \sum_i x_{ij} = 1 \quad i = 1, \dots, 10 \end{aligned}$$

The first set of constraints specifies that each woman is paired with only one man, the second that each man is paired with only one woman. The simple structure of the problem ensures that in any basic solution every variable  $x_{ij}$  will have the value 0 or 1.

Figure 1 shows a SESAME/DATAMAT session in which this problem is formulated and solved. Input typed at the terminal is printed in lower case letters, while output from the system appears in capitals. Greater-than signs (>) on input lines are prompts from DATAMAT.

After SESAME is invoked and DATAMAT is called, a TABLE verb (1) is used to create a table that holds the matrix  $[a_{ij}]$  of ratings. Each row is labelled with a woman's name, each column with a man's. Subsequently, an ENFILE verb (2) stores the table on a permanent file (from which it can be recalled if it is needed again), and a DISPLAY verb prints the table's contents for inspection.

Generation of an LP model for the problem begins with the verb NEWMODEL (4). The rows of the model are defined with three ROW commands (5-7), which also implicitly define the columns. The objective row (5) is named OBJ; the rows that limit each woman to one man (6) are named with the women's names; and the rows that constrain the men (7) are given the men's names. The columns for the variables  $x_{ij}$  are named by concatenating the first four letters of the women's names with the first four letters of the men's names. (The function MASK and the concatenation operator & are used for this purpose.)

40  
100

Figure 1. A SESAME/DATAMAT session that generates and solves a simple assignment problem (Example 1).

```

sesame
SESAME V9.2
SESAME COMMAND: call datamat
ALL FILES ALREADY CLOSED
CORE WAS NOT SET UP
> table g:ratings = bill, john, egbert, cuthbert, joe, gaston,
> chauncey, clyde, newt, waldo
> joe = 9,6,3,0,2,8,7,4,1,5
> mary = 3,7,8,2,1,0,5,4,0,6
> chloe = 4,2,1,6,0,8,3,9,7,5
> beulah = 6,3,5,7,9,0,1,4,2,8
> phoebe = 7,5,6,9,1,8,3,0,2,4
> octavia = 1,10,8,4,5,3,6,9,2,7
> juliet = 6,8,10,9,4,3,5,1,7,2
> myrtle = 7,8,4,3,2,6,1,9,5,0
> elga = 3,9,4,2,5,6,7,0,8,1
> mabel = 9,3,1,8,0,4,2,7,6,5
> .....
> enfile table as g:ratings
1 TABLES ENFILED
> display g:ratings
G:RATINGS = BILL NEWT JOHN WALDO EGBERT CUTHBERT JOE GASTON CHAUNCEY CLYDE
JANE = 9,0000000, 6,0000000, 3,0000000, 8,0000000, 2,0000000, 1,0000000, 8,0000000, 7,0000000, 4,0000000,
MARY = 3,0000000, 7,0000000, 8,0000000, 2,0000000, 6,0000000, 1,0000000, 5,0000000, 9,0000000, 4,0000000,
CHLOE = 4,0000000, 2,0000000, 1,0000000, 6,0000000, 3,0000000, 8,0000000, 7,0000000, 9,0000000, 5,0000000,
BEULAH = 6,0000000, 3,0000000, 5,0000000, 7,0000000, 9,0000000, 2,0000000, 1,0000000, 8,0000000, 4,0000000,
PHOEBE = 7,0000000, 5,0000000, 6,0000000, 9,0000000, 1,0000000, 8,0000000, 3,0000000, 7,0000000, 9,0000000,
OCTAVIA = 1,0000000, 10,0000000, 8,0000000, 4,0000000, 5,0000000, 6,0000000, 3,0000000, 9,0000000, 7,0000000,
JULIET = 6,0000000, 8,0000000, 10,0000000, 9,0000000, 4,0000000, 3,0000000, 5,0000000, 1,0000000, 9,0000000,
MYRTLE = 7,0000000, 8,0000000, 4,0000000, 3,0000000, 2,0000000, 6,0000000, 1,0000000, 9,0000000, 7,0000000,
ELGA = 3,0000000, 9,0000000, 4,0000000, 2,0000000, 5,0000000, 7,0000000, 8,0000000, 6,0000000, 1,0000000,
MABEL = 9,0000000, 3,0000000, 1,0000000, 8,0000000, 5,0000000, 6,0000000, 4,0000000, 2,0000000, 7,0000000,

```

```

> newmodel
> row obj, (mask(gratings(12,0),'****0000')) &
(C) gratings(0,1) = gratings(12,11)
> row gratings(11,0) <eqtype>, (mask(gratings(12,11) &
(C) gratings(0,12) = 1
> row gratings(0,11) <eqtype>, (mask(gratings(12,0),'****0000')) &
(C) gratings(0,11) = 1
> rhs rhs, gratings(11,0) = 1, gratings(0,11) = 1
> enfle model as pairings
      ROWS      COLUMNS      RHS      RANGES      BOUNDS      GUB-S      STR COEFS DENSITY      INDIRECT
      21      100      1      0      0      0      291      .13857143 0
> quit
(10) leave DATAMAT

```

TERMINATING DATAMAT, RETURN TO SESAME  
QUIT

SESAME COMMAND: call setup max smodel=pairings

SESAME COMMAND: call iterate subj=obj srhs=rhs sflog=0 sfinvert=200

PLEASE SOLUTION

OPTIMAL SOLUTION

SESAME COMMAND: call solution inlgls = obj s active

SOLUTION

OPTIMAL SOLUTION AT ITERATION NUMBER 43

...NAME...	...ACTIVITY...	DEFINED AS
FUNCTIONAL	85.000000	OBJ
RESTRAINTS		RHS

ROWS SECTION

NUMBER	...ROW...	AT	...ACTIVITY...	SLACK ACTIVITY	...LOWER LIMIT...	...UPPER LIMIT...	DUAL ACTIVITY
1	OBJ	BS	85.000000	-85.000000	NONE	NONE	-1.0000000

COLUMNS SECTION

NUMBER	COLUMN	AT	...ACTIVITY...	...INPUT COST...	...LOWER LIMIT...	...UPPER LIMIT...	REDUCED COST
31	MAREBILL	BS	1.0000000	9.0000000	.	NONE	.
37	OCTA JOHN	BS	1.0000000	10.000000	.	NONE	.
43	JULIECBE	BS	1.0000000	10.000000	.	NONE	.
50	PIRECUITH	BS	1.0000000	9.0000000	.	NONE	.
65	REULIOE	BS	1.0000000	9.0000000	.	NONE	.
74	CHLOFAST	BS	1.0000000	8.0000000	.	NONE	.
82	JANECHAU	BS	1.0000000	7.0000000	.	NONE	.
93	MYTHICLYD	BS	1.0000000	9.0000000	.	NONE	.
110	OLGANENT	BS	1.0000000	8.0000000	.	NONE	.
113	MARYWALD	BS	1.0000000	6.0000000	.	NONE	.

SESAME COMMAND:

The special expressions !1 and !2 create implicit loops through all rows or columns of the table. Expressions of the form

G:RATINGS(!n,0)

loop through all women's names in the table; ones of the form

G:RATINGS(0,!n)

loop through all men's names; and the expression in (5):

G:RATINGS(!1,!2)

loops through all the numerical elements of the table, !1 creating an outer loop over the rows and !2 creating an inner loop over the columns.

A right-hand side vector, named RHS, is next defined by use of an RHS verb (8), and the model is complete. It is stored on a permanent model file with the ENFILE verb (9), and the QUIT verb (10) returns control to the main SESAME environment.

It is now a straightforward task to set the model up (11), solve it (12), and display the active variables (13). From the variables' names the actual pairings are easily deduced.

## EXAMPLE 2. A GENERAL INPUT-OUTPUT MODEL

### (a) The problem

An economy comprises a variety of industries, each manufacturing a particular product. Production is to be modeled over a number of time periods, subject to the following constraints:

There is an initial stock of each product. Stocks may be built up or run down in subsequent periods.

Each industry requires certain fixed amounts of various inputs for each unit of its product manufactured. The inputs are of two sorts: *endogenous* inputs which are products of industries in the economy, and *exogenous* inputs whose supplies are postulated (labor, for instance).

Each industry has an initial capacity. Capacities may be increased (but not decreased) in any period, but the added capacity may not be used until the following period. Analogously to production, each industry requires certain fixed amounts of various inputs - endogenous and exogenous - for each unit of increase in capacity.

There is an initial supply of each exogenous input; the supply increases by a fixed percentage in each subsequent period.

Each industry must satisfy an exogenous demand for its product in each period. There is an initial exogenous demand for each product, and this demand increases by a fixed percentage in each subsequent period.

The objective is to maximize the total production of one particular activity over all periods.

To express the problem as a linear program, it must first be converted to a more precise notation. Let  $T$  be the number of periods,  $n$  the number of industries, and  $\hat{n}$  the number of exogenous inputs. The variables may then be specified as:

- $s_i(t)$  stock of product  $i$  at beginning of period  $t$ ;  
 $i = 1, \dots, n; t = 1, \dots, T+1$
- $x_i(t)$  quantity of product  $i$  manufactured in period  $t$ ;  
 $i = 1, \dots, n; t = 1, \dots, T$
- $r_i(t)$  increase in capacity of industry  $i$  in period  $t$ ;  
 $i = 1, \dots, n; t = 1, \dots, T$

The parameters of the problem can be specified as four matrices and six vectors, whose elements are:

- $A_{ij}$  number of units of product  $i$  required to produce 1 unit of product  $j$ ;  $i = 1, \dots, n; j = 1, \dots, n$
- $\hat{A}_{ij}$  number of units of exogenous input  $i$  required to produce 1 unit of product  $j$ ;  $i = 1, \dots, \hat{n}; j = 1, \dots, n$
- $D_{ij}$  number of units of product  $i$  required to increase the capacity of industry  $j$  by 1 unit;  $i = 1, \dots, n; j = 1, \dots, n$
- $\hat{D}_{ij}$  number of units of exogenous input  $i$  required to increase the capacity of industry  $j$  by 1 unit;  $i = 1, \dots, \hat{n}; j = 1, \dots, n$
- $e_i$  initial stock of product  $i$ ;  $i = 1, \dots, n$
- $c_i$  initial capacity of industry  $i$ ;  $i = 1, \dots, n$
- $\hat{c}_i$  initial supply of exogenous input  $i$ ;  $i = 1, \dots, \hat{n}$
- $\gamma_i$  fractional increase in supply of exogenous input  $i$  per period (1/100 of percentage increase);  
 $i = 1, \dots, \hat{n}$

10  
62



- $b_i$  initial exogenous demand for product  $i$ ;  $i = 1, \dots, n$
- $\beta_i$  fractional increase in exogenous demand for product  $i$  per period;  $i = 1, \dots, n$

The objective is now to maximize the total production,  $\sum_{\tau} x_z(\tau)$ , of a chosen industry  $z$ . The constraints may be expressed in five classes. First are the *initial stock constraints*

$$s_i(1) = e_i \quad i = 1, \dots, n$$

Second are the *production constraints*, which specify that the quantity of a product manufactured in a period equals (i) the quantity required by all industries for production in the period, plus (ii) the quantity required by all industries for expansion of capacity in the period, plus (iii) the exogenous demand in the period, plus (iv) the net change in stocks:

$$x_i(t) = \sum_j A_{ij} x_j(t) + \sum_j D_{ij} r_j(t) + \beta_i^{t-1} b_i + s_i(t+1) - s_i(t) \\ i = 1, \dots, n; t = 1, \dots, T$$

Third, *capacity constraints* dictate that production must not exceed an industry's capacity, which is its initial capacity plus the sum of all increased in prior periods:

$$x_i(t) \leq c_i + \sum_{\tau=1}^{t-1} r_i(\tau) \quad i = 1, \dots, n; t = 1, \dots, T$$

Fourth, *supply constraints* ensure that the quantity of exogenous inputs consumed does not exceed the available supplies:

$$\sum_j \hat{A}_{ij} x_j(t) + \sum_j \hat{D}_{ij} r_j(t) \leq \gamma_i^{t-1} \hat{c}_i \quad i = 1, \dots, \hat{n}; t = 1, \dots, T$$

Finally, all variables must be non-negative.

Moving variables to the left-hand side, the entire LP problem is formulated thus ( $I$  representing the  $n$ -by- $n$  identity matrix):

- (1)  $s_i(1) = e_i \quad i = 1, \dots, n$
- (2)  $\sum_j (I-A)_{ij} x_j(t) - \sum_j D_{ij} r_j(t) - s_i(t+1) + s_i(t) = \beta_i^{t-1} b_i \\ i = 1, \dots, n; t = 1, \dots, T$
- (3)  $x_i(t) - \sum_{\tau=1}^{t-1} r_i(\tau) \leq c_i \quad i = 1, \dots, n; t = 1, \dots, T$

$$\begin{aligned}
 (4) \quad & \sum_j \hat{A}_{ij} x_j(t) + \sum_j \hat{D}_{ij} r_j(t) \leq \gamma_i^{t-1} \hat{c}_i \\
 & i = 1, \dots, \hat{n}; t = 1, \dots, T \\
 (5) \quad & s_i(t) \geq 0 \\
 & i = 1, \dots, n; t = 1, \dots, T+1 \\
 (6) \quad & x_i(t) \geq 0 \\
 & i = 1, \dots, n; t = 1, \dots, T \\
 (7) \quad & r_i(t) \geq 0 \\
 & i = 1, \dots, n; t = 1, \dots, T
 \end{aligned}$$

### (b) Data Tables

A configuration of *tables* for the problem data is now decided upon, and these tables are represented as a deck on a card-image file which will be read by DATAMAT.\* DATAMAT tables are basically two-dimensional arrays of numerical or character values; a table thus serves to hold one matrix or one or more vectors. Each table also contains an extra column of *stubs*, which are names that identify the rows of the table, and an extra row of *heads* which name the columns of the table. Proper choice of stubs and heads is essential to problem generation, since they are concatenated in forming model terminology (see below).

For the present problem, four tables are required to hold the four matrices of parameters:

<u>Matrix</u>	<u>Table name</u>
A	G:A
$\hat{A}$	G:AX
D	G:D
$\hat{D}$	G:DX

The stubs of these tables represent inputs - endogenous (G:A, G:D) or exogenous (G:AX, G:DX) - while the heads always represent endogenous products. (The "G:" at the beginning of each table name indicates that its elements are numerical values.) Four tables are also used for the six vectors of parameters:

\*Card-image files are easily created and maintained by use of the CMS context editor. To learn more, consult *IBM Virtual Machine Facility/370: EDIT Guide* (GC20-1805).

10  
36

<u>Vector(s)</u>	<u>Table name</u>
$e$	G:E
$c$	G:C
$\hat{c}, \gamma$	G:CX
$b, \beta$	G:B

The stubs again represent either products (G:E, G:C, G:B) or exogenous inputs (G:CX); the heads identify the particular vectors.

Two additional tables, M:EN and M:EX, serve as indexes to the heads and stubs. The stubs of M:EN are the ones that identify endogenous products, and the stubs of M:EX are the ones that correspond to exogenous inputs. The body of each table is a single vector of character strings each of which identifies the corresponding stub in a bit more detail. (The prefix "M:" indicates tables whose elements are strings of up to 8 characters.)

Figure 2. A data-table deck for a three-industry instance of the input-output model (Example 2).

```

NAME      GROWTABS
TABLE G:A = AI, EL, ME
          AI = 0, 0, 0
          EL = 3, 0.1, 0.05
          ME = 2, 0.4, 0.1

.....
TABLE G:AX = AI, EL, ME
          LA = 500, 25, 0.2

.....
TABLE G:D = AI, EL, ME
          AI = 0, 0, 0
          EL = 2, 3, 0.5
          ME = 5, 1, 2

.....
TABLE G:DX = AI, EL, ME
          LA = 7, 2, 0.5

.....
TABLE G:E = E
          AI = 0
          EL = 100
          ME = 5000

.....
TABLE G:C = C
          AI = 100
          EL = 13500
          ME = 1E5

.....
TABLE G:CX = CX, PCT
          LA = 1E6, 1.1

.....
TABLE G:B = B, PCT
          AI = 0, 0
          EL = 3000, 1.1
          ME = 6E4, 1.05

.....
TABLE M:EN = EN
          AI = AIRPLANE
          EL = ELSC
          ME = METALS

.....
TABLE M:EX = EX
          LA = LABOR

.....
ENDATA

```

A typical deck for an economy with three industries and one exogenous input is shown in Figure 2. The endogenous product stubs (and heads) are AI (airplanes), EL (electronics), and ME (metals). The exogenous input stub is LA (labor). Airplanes are intended as the product whose production is to be maximized: note that they are the only output that does not serve as an input to other industries.

(c) Model terminology

The model's *rows* (constraints) and *columns* (variables) must all be assigned names in some consistent way. These names are formed by concatenation of table stubs and other identifiers.

Column names for the present problem are concatenations of three elements. They are of the forms:

$SnnTt$

$XnnTt$

$RnnTt$

where the first character indicates the type of variable - stock (S), quantity of production (X), or increase in capacity (R) -  $nn$  is the stub representing some endogenous product, and  $t$  is a period number. Using the tables from Figure 2, for instance, variable SAIT2 is the stock of airplanes at the beginning of period 2; XMET5 is the quantity of metals produced in period 5.

Row names are handled similarly. Letting  $nn$  and  $xx$  be the stubs for endogenous and exogenous goods, respectively, and letting  $t$  be a period number, the forms are:

<u>Constraint</u>	<u>Form of row name</u>
Production (2)	PRnnTt
Capacity (3)	CAnnTt
Supply (4)	SUxxTt

Again using Figure 2, row PRELT4 specifies the production constraint on electronics in period 4; SULAT3 gives the constraint imposed by the supply of labor in period 3. (The initial stock constraints (1) and the non-negativity constraints (5-7) are not modeled explicitly as rows: see below.)

12  
82

The objective row is also given a name, of the form MAX $zz$  where  $zz$  is the stub representing the activity that is to be maximized. In the example, airplane production is the objective, so the objective row name is MAXAI.

(d) A macro for the problem

Shown in Figure 3 is a DATAMAT program - called a *macro* - that can generate a model for any specific instance of the problem we have specified. The industries and exogenous input to be modeled are determined entirely by the construction of the data tables. Other information is taken from parameters to the macro: the first parameter is a short identifier used to form the name of the model when it is enfiled, the second gives the number of periods, and the third is the stub abbreviation for the industry whose production is to be maximized.

Macros are stored as decks on a special card-image file called a *maclib*. Each macro begins with a NAME card that gives the name by which it is invoked - GROWTH in the present case. The last card, ENDATA, indicates the end of the macro. Intervening cards comprise a sequence of DATAMAT commands, or *verbs*, which are executed each time you call the macro.

For clarity, the macro has been divided into functional sections, each identified by a heading marked with an asterisk (\*) in column 1; all lines so marked are interpreted as comments and are thus not processed as DATAMAT verbs. The commands within each section are also accompanied by comments; DATAMAT reads only the first 72 characters of each macro line, so these comments are begun in column 73. Further commentary on each section follows (numbers in parentheses are comment line numbers):

Process parameters.

The parameters to the macro are automatically given the special names %1, %2, and %3. The latter two, whose values are used in several places, are here assigned to local variables (I:PERIODS and N:OBJ, respectively) whose names have some mnemonic significance.

Create index table for periods.

Here a FOKM verb creates a table consisting only of stubs, of the form Tt where  $t$  ranges from 1 to the number of periods. This table is used to

regulate two loops in the macro, and the stubs are employed in forming row and column names.

Create table equal to  $(I - A)$ .

A table G:IMINUSA, identical in form to G:A, is created to hold the matrix  $(I - A)$  employed in the production constraints.

Note here the *implicit* use of loop indices !1 and !2. These automatically create one-statement loops through all stubs or heads of the tables G:IMINUSA and G:A. Where both !1 and !2 appear (3), the former creates an outer loop and the latter an inner one.

Move initial exogenous supplies and demands to work areas.

The macro must increase the values for exogenous supplies and demands by a fixed percentage in each time period. A table G:CURCX is created to hold the current exogenous supplies (1) and its elements are set to the initial exogenous supply values specified in G:GX (2). Similarly, G:CURB is created and initialized for the exogenous demands values (3-4).

Specify objective row.

The NEWMODEL verb (1) indicates that generation of a new model is to begin. The ROW verb following (2) defines the objective row, and specifies its coefficients in various columns. Columns are defined automatically when they are first referred to (although a COL statement is available to define columns explicitly when necessary).

Note the use of an ampersand (&) as a concatenation operator to form row and column names.

Specify bounds on initial stock variables.

A bound set, INITS, is defined. It specifies the initial stock constraints by fixing the value of every first-period stock variable to the quantity of initial stocks specified in table G:E. (All other variables, which are not explicitly bounded or fixed, are assumed by SESAME to be non-negative.)

Main loop.

A LOOP statement causes the index !1 to be stepped explicitly through each of the stubs of G:T. Since G:T was created with a stub Tt for each period t, this loop is executed exactly once for each period. Each pass through the loop defines constraint rows for one period.

18  
12

Figure 3. A DATAMAT macro to generate any instance of the input-output model (Example 2).

```

NAME      GROWTH
*
*
* PROCESS PARAMETERS
*
*   CALC I*PERIODS = X2
*   M*HIP N*OBJ = X3
*
*
* CREATE INDEX TABLE FOR TIME PERIODS
*
*   FORM M*T = I & I*PERIODS
*
*
* CREATE TABLE EQUAL TO I - A
*
*   FORM G*MINUSA = G*A(STUB), G*A(HEAD)
*   CALC G*MINUSA(11,11) = 1
*   CALC G*MINUSA(11,12) = G*MINUSA(11,12) - G*A(11,12)
*
*
* MOVE INITIAL EXOGENOUS SUPPLIES AND DEMANDS TO WORK AREAS
*
*   FORM G*CURCX = G*CX(STUB), CX&1
*   CALC G*CURCX(11,CX1) = G*CX(11,CX)
*   FORM G*CURB = G*B(STUB), B&1
*   CALC G*CURB(11,B1) = G*B(11,B)
*
*
* SPECIFY OBJECTIVE ROW
*
*   NEWMODEL
*   ROW MAX & N*OBJ, X & N*OBJ & M*T(11,0) = 1
*
*
* SPECIFY BOUNDS ON INITIAL STOCK VARIABLES
*
*   BOUND INITS, S & M*EN(11,0) & M*T(1,0) = G*E(11,E)
*
*
* MAIN LOOP — ONE PASS FOR EACH TIME PERIOD
*
*   LOOP M*1(11,0) <NE> DUMMY
*

```

(1) 2ND PARAMETER — NUMBER OF PERIODS  
(2) 3RD PAR. — ABBREV. FOR OBJ. INDUSTRY

(1) FORM INDEX TABLE M\*T

(1) FORM TABLE FOR I - A  
(2) SET TABLE TO 1  
(3) SUBTRACT G\*A FROM TABLE TO GET I - A

(1) I FORM AND INITIALIZE TABLE OF  
(2) I CURRENT EXOGENOUS SUPPLIES  
(3) I FORM AND INITIALIZE TABLE OF  
(4) I CURRENT EXOGENOUS DEMANDS

(1) START GENERATING MODEL  
(2) DEFINE OBJECTIVE ROW

(1) DEFINE BOUND SET TO FIX INITIAL STOCKS

(1) LOOP OVER ALL STUBS IN INDEX TABLE M\*T

\* \* SPECIFY EXOGENOUS SUPPLY CONSTRAINTS FOR PERIOD

```

LOOP M:EX(12,0) <NE> DUMMY
  MANIP N:SU = SU & M:EX(12,0) & M:T(11,0)
  ROW N:SU, X & M:EN(13,0) & M:T(11,0) = G:AX(12,13)
  ROW N:SU, R & M:EN(13,0) & M:T(11,0) = G:DX(12,13)
  RHS RHS, N:SU <LETYPE> = G:CURCX(12,CX1)
CONTINUE

```

\* \* \*

\* \* SPECIFY CAPACITY CONSTRAINTS FOR PERIOD

```

LOOP M:EN(12,0) <NE> DUMMY
  MANIP N:CA = CA & M:EN(12,0) & M:T(11,0)
  ROW N:CA, X & M:EN(12,0) & M:T(11,0) = 1
  LOOP M:T(13,0) <LT> M:T(11,0)
    ROW N:CA, R & M:EN(12,0) & M:T(13,0) = -1
  CONTINUE
  RHS RHS, N:CA <LETYPE> = G:C(12,C)
CONTINUE

```

\* \* \*

\* \* SPECIFY PRODUCTION CONSTRAINTS FOR PERIOD

```

LOOP M:EN(12,0) <NE> DUMMY
  MANIP N:PR = PR & M:EN(12,0) & M:T(11,0)
  ROW N:PR, X & G:IMINUSA(0,13) & M:T(11,0) = G:IMINUSA(12,13)
  ROW N:PR, R & G:D(0,13) & M:T(11,0) = -G:D(12,13)
  ROW N:PR, S & M:EN(12,0) & M:T(11,0) = 1
  ROW N:PR, S & M:EN(12,0) & BUMP(M:T(11,0)) = -1
  RHS RHS, N:PR <EQTYPE> = G:CURB(12,B1)
CONTINUE

```

\* \* \*

\* \* UPDATE EXOGENOUS SUPPLIES AND DEMANDS FOR NEXT PERIOD

```

IF M:T(11,0) <NE> M:T(1:PERIODS,0), 1
GOTO SKIP
  CALC G:CURB(12,B1) = G:CURB(12,B1) * G:B(12,PCT)
  CALC G:CURCX(12,CX1) = G:CURCX(12,CX1) * G:CX(12,PCT)

```

//SKIP

\* \* \*

\* \* END OF MAIN LOOP

CONTINUE

\* \* \*

\* \* ENFILE MODEL

```

MANIP N:NAME = X1 & SHIFT(MASK(M:T(1:PERIODS,0),0),1) & N:OBJ
ENFILE MODEL AS N:NAME

```

\* \* \*

ENDATA

(1) LOOP OVER STUBS IN EX.-INPUT TABLE M:EX  
(2) SET N:SU TO NAME OF SUPPLY-CONSTRAINT ROW  
(3) SET PRODUCTION-VARIABLE COEFFS. ON ROW  
(4) SET CAPACITY-VARIABLE COEFFS. ON ROW  
(5) SET RHS FOR ROW TO CURRENT SUPPLY  
(6) STEP 12 AND LOOP

(1) LOOP OVER STUBS IN END.-PROD. TABLE M:EN  
(2) SET N:CA TO NAME OF CAP.-CONSTRAINT ROW  
(3) SET COEFF. OF 1 FOR PRODUCTION VARIABLE  
(4) LOOP OVER ALL PREVIOUS TIME PERIODS  
(5) SET COEFFS. FOR CAPACITY VARIABLES  
(6) STEP 13 AND LOOP  
(7) SET RHS FOR ROW TO INITIAL CAPACITY  
(8) STEP 12 AND LOOP

(1) LOOP OVER STUBS IN END.-PROD. TABLE M:EN  
(2) SET N:PR TO NAME OF PROD.-CONSTRAINT ROW  
(3) SET COEFFS. FOR PRODUCTION VARIABLES  
(4) SET COEFFS. FOR CAPACITY VARIABLES  
(5) SET COEFF. ON CURRENT-PERIOD STOCK VAR.  
(6) SET COEFF. ON NEXT-PERIOD STOCK VAR.  
(7) SET RHS FOR ROW TO CURRENT EXOG. DEMAND  
(8) STEP 12 AND LOOP

(1) CHECK IF CURRENT PERIOD IS LAST  
(2) IF YES, SKIP THIS PART  
(3) IF NO, UPDATE EXOGENOUS DEMANDS  
(4) UPDATE EXOGENOUS SUPPLIES  
(5)

(1) STEP 11 AND LOOP

(1) CREATE MODEL NAME  
(2) WRITE MODEL TO MODEL FILE



Specify exogenous supply constraints.

ROW verbs (3-4) define and specify coefficients on the supply constraint rows for the current period. A right-hand-side vector named RHS is defined and specified with the RHS verb (5). The RHS verb is also used here to specify the row constraint type, LETYPE, which indicates that the row sum must be less than or equal to the right-hand side value. (Row type can also be specified with a ROW verb. Rows not assigned a type - such as the objective row - are assumed unconstrained.)

Note the occurrence here of nested loops: an inner LOOP statement varies the index !2, and that loop in turn contains implicit loops (3-4) that vary !3.

Specify capacity constraints.

Again ROW (3, 5) and RHS (7) verbs define constraints, this time for capacity. Here LOOP statements are nested to a depth of 3; but the innermost loop is specified so that !3 is set only to stubs of G:T that correspond to periods prior to the current one.

Specify production constraints.

This is similar to the generation of the other constraints. Note use of the function EUMF (6) to create the name of the stock variable for the period after the current one.

Update exogenous supplies and demands.

If the main loop has yet to reach the final period, exogenous supplies and demands (stored in G:CURCX and G:CURB, respectively) are increased by the specified ratios (3-4). A combination of an IF statement (1) and a GOTO statement (2) is used to skip the updating in the final period. The expression //SKIP is a label to which control is transferred by the GOTO.

End of main loop.

The index !1 is stepped, and the loop is repeated for the following time period. When all periods are accounted for, the loop is finished and control passes to the following statement.

Enfile model.

The SHIFT and MASK functions are used (1) to create a name for the model of the form *mmmmtrn*, where *mmmm* is the identifier given as the first parameter, *t* is the number of periods, and *m* is the stub abbreviation of the industry whose production is to be maximized. An ENFILE verb (2) then places the completed model on a model file under the given name.

(e) Generating a model

Figure 4 shows the beginning of a typical SESAME/DATAMAT session in which a model is generated by use of the macro of Figure 3. The problem is to be modeled for ten periods, with three industries and one exogenous input as specified in the data tables of Figure 2. Lines typed from the terminal are shown in lower case letters, while responses from the system are in capitals; greater-than signs on input lines are prompting characters also typed by the system.

Figure 4. A DATAMAT session that generates the input-output model from the tables of Figure 2, using the macro of Figure 3.

```

sesame
SESAME V9.2
SESAME COMMAND: >call datamat
ALL FILES ALREADY CLOSED
CORE WAS NOT SET UP
> set maclib = growth
> readtab m*tabs, growtabs growtabs
> display m*tabs
M* TABS      =TYPE
  A          =GN
  AX         =GN
  D          =GN
  DX         =GN
  E          =GN
  C          =GN
  CX         =GN
  B          =GN
  EN         =MN
  EX         =MN
> growth exam 10 al
EXAM10AI REPLACES EXISTING MODEL IN FILE
  ROWS      COLUMNS  RHS      RANGES  BOUNDS  GUB-S  STR COEFS DENSITY  INDIRECT
  71         23         1         0         1         0      425      .06436466  0
> quit

TERMINATING DATAMAT. RETURN TO SESAME
QUIT

SESAME COMMAND: call setup max $model=exam10ai $rhs=rhs $obj=maxai $bound=initls
SESAME COMMAND:

```

The session begins with the invocation of SESAME and the calling of DATAMAT. The first DATAMAT command, SET MACLIB, declares the name of the maclib file on which the macro of Figure 3 resides. There follows a READTAB verb that reads into working storage, from a file GROWTABS and a deck of the same name, the tables of Figure 2. READTAB also creates a table M:TABS whose stubs are the names of the tables read; the following verb, DISPLAY, prints the contents of M:TABS at the terminal.

The macro GROWTH is now invoked. Its parameters are EXAM, the identifier used to form the model name; 10, the number of periods; and AI, the abbreviation for the airplane industry, whose production is to be maximized. The full name of the generated model is thus EXAM10AI. A message indicates that an old model by this name previously resided on the model file, but was deleted in favor of the new model. Component counts for the enfiled model are then printed, and the macro concludes.

QUIT next returns execution to the SESAME environment, where the new model is set up by use of the SETUP procedure. It is now ready to be solved by calling ITERATE (not shown).

## APPENDIX. OVERVIEW

DATAMAT is a data management system specialized to mathematical programming. It is implemented as a SESAME procedure, and is thus invoked (in the manner of other procedures) by typing CALL DATAMAT followed optionally by arguments. DATAMAT is conceptually independent of the other SESAME procedures, however, and employs an extensive command language of approximately 55 verbs. It is thus best regarded as a system in itself, but one which - through common use of the communication region and internal model, map, and result files - is compatible with and dependent on SESAME.

DATAMAT provides a number of major capabilities but should not be regarded as equivalent to any one or all of them. Among these capabilities are:

- Generation and maintenance of basic data
- Model generation
- Model revision
- Report generation
- Grand cycle calculations and control
- Ad hoc calculations
- Inspection and display of various quantities

Before starting on a description of DATAMAT and an example of its use, an orientation section will be presented.

### 1. Conceptual Orientation

One learns in high school algebra to abstract numerical attributes of real things or situations and to represent them with letters. This technique is progressively elaborated to include subscripts, vector

AE  
ER

notation, matrices, superscripts, and so forth. By the time one is discussing LP algorithms, a mere notation such as

$$\sum_j a_{ij} X_j \leq b_i, \quad i=1, \dots, m$$

or even

$$A X \leq b$$

is deemed sufficient to denote a possibly very large array with various special conditions. We often neglect to even mention  $X \geq 0$ , it being assumed this is a standard condition.

If one is formulating an actual LP model, then he is inclined to elaborate the notation symbolically, such as

$$LCON(i,t): -1.*LUN(i,t) + \sum_{j \in S3} LTAB(i,j,t) * X(j,t) = LA(i,t)$$


It is silly to try to explain, in general, what such notation means. It is dependent on pages of discussion about a particular problem and incorporates the writer's own mnemonics. It is certainly not silly to use such notation but one should remain aware that it is a mixture of mathematical notation and abbreviated symbology. Such expressions are not "proper" in any classical sense since they involve both numerical representations and implied identifiers. The identifiers are supposed to be concatenated in various combinations to account for all activities and constraints in the model.

An LP model is largely combinatorial in nature. Typically, there are classes of constraint rows and classes of LP variables which intersect only selectively. This is the cause of the well-known sparseness of LP model

matrices, on which both solution algorithms and associated data management procedures depend heavily. The actual LP coefficients, i.e., matrix elements are frequently obtained from basic data tables which are themselves dense, and relatively small. The amount of information contained in an LP model is only secondarily derived from actual numerical data; the larger part is in the identification of the various ways in which these data are related. It is possible and even usual to formulate an LP model before the exact dimensions and numerical values of the basic data tables are known. One must, of course, know the classes of information and the overall logic of the situation being modeled.

The above considerations indicate the importance of LP identifiers, i.e., row and column names. The creation of a complete large LP model is a tedious process and virtually impossible to do by hand without error. Tailor-made matrix generators are frequently programmed--often in FORTRAN--for a particular set of models. However, this requires the services of a computer programmer and neither FORTRAN nor any other widely-available language is well suited to the task.

What is needed is a reasonably general language and processor which can deal with numerical values and symbology in a coordinated fashion. This is one of the main features and purposes of DATAMAT. However, even when restricted to the field of mathematical programming models - even further to LP models with various extensions - a language requires considerable generality. This results in some amount of specialized syntax and DATAMAT reflects this. However provisions are made for saving intricate statements which have multiple uses and invoking them as needed with substitutable arguments. Such canned strings of generalized statements are called macros and they constitute an important capability in the language.



To use a language such as DATAMAT, one must pay a great deal of attention to organization of basic data and its identifiers. Once this has been done, it is easy to create combinatorial LP identifiers by concatenating basic data identifiers. Very little attention need be paid to numerical values in the planning stage - they are readily altered as required. However, a serious error in basic terminology may lead to considerable rework of basic tables. Although DATAMAT includes statement forms to manipulate, replicate and modify basic data structures, the implications of such alterations may require study and rechecking.

The term data tables has already been used and, as might be supposed, they are at the heart of the use of DATAMAT. A table, as used here, is a rectangular array of values whose rows and columns have symbolic identifiers. Elements in a table may be either numeric or symbolic, and may be referenced either by numerical indices or by identifiers or a mixture. Furthermore, special index flags allow one to run over all elements in a row or a column, or over all identifiers which match those of another table, regardless of order and in a direct or transpose fashion. These flags are represented by special characters and may give some statements a rather bizarre appearance. Once one becomes familiar with this shorthand notation, however, they seem perfectly natural -- and save considerable typing.

Tables are of three kinds, called G-tables, M-tables and H-tables. The elements of G-tables are REAL\*8, the elements of M-tables are ALPHA\*8. H-table elements are also ALPHA\*8 but are regarded as strings of characters in multiples of 8.

All table row names (stubs) and column names (heads) are ALPHA\*8 and may be used in symbolic manipulations. Normally, however, only one to four nonblank characters are used for the stubs and heads of most tables. Since many tables are used to generate LP submatrices -- by various types of expansion -- it is usually necessary to concatenate the stub and head symbols with other identifier-parts to create meaningful and unique LP identifiers.

LP identifiers, and to a lesser extent table stubs and heads, should not be true mnemonics or acronyms in most cases. Rather, they should be regarded as encodings. These encodings can, if necessary, be further associated with true mnemonics and even short readable text through the use of coordinated M- and H-tables. The identifiers for an LP problem with, say, 1000 rows and 2000 columns have a highly combinatorial nature and it is impossible to condense concatenated mnemonics of any value into 8 characters. Indeed, in many applications, LP identifiers are of no real interest anyway, unless it becomes necessary to debug the model. The use of DATAMAT helps to bypass the necessity for interpreting cryptic LP identifiers and this should be taken advantage of.

DATAMAT deals with 26 kinds of entities, including tables, plus SESAME CR cells. Several different kinds of entities may be referenced in one statement, i.e., with one verb. To keep these sorted out and to avoid the problems of ambiguity, each kind of entity is identified with a single-letter prefix attached to the referent with a full colon. The full colon is preassigned in DATAMAT for this purpose.

Am  
82



The 26 kinds of entities may be grouped in 8 classes:

Local variables	E:, I:, N:, O:, Q:
Tables	G:, H:, M:
Model components	A:, B:
LP results	C:, D:, P:, R: thru Z:
FORTTRAN (or other) functions	F:
Built-in functions	K:
FORTTRAN-style arrays	L:
SESAME general maps	J:

The choice of letters favor LP models and results (the latter being by far the most extensive class) and the remaining letters were assigned in the best way possible.

References to CR cells are recognized by the leading dollar sign as in SESAME. The single quote (') is absolutely preempted to enclose non-standard character strings, except in FORMAT statements where single quotes are treated as in FORTRAN.

There are only three flags: the exclamation point (!) for automatic table indexing; the double quote (") for table head and stub name-matching; and the percent sign (%) for substitutable macro arguments. The only remaining unusual characters are the semi-colon (;) which denotes end of useful information (so a comment can follow), and the left and right angle brackets (<, >), which are used to enclose relational and boolean operators and for certain special purposes in defining LP model components.

The four standard arithmetic operators and the equal sign are used in normal fashion, i.e.,

=	right to left replacement
+	addition
-	subtraction or leading minus sign
*	multiplication
/	division

The vertical bar is also recognized for absolute value. Exponentiation is provided through built-in K:functions.

This symbology and notation is used to form expressions and phrases\*. For example, the phrase

$$E:ANS = I:ARG * 3.0$$

denotes multiplication of an integer variable (I:ARG) by the literal number 3.0 with the result to become the value of the real variable E:ANS.

Such expressions and phrases are never used alone but must be preceded by an appropriate verb, in the above case CALC. Hence the full statement would be

$$CALC \ E:ANS = I:ARG * 3.0$$


---

\* A phrase is a construction of the form "result = expression".

10  
102

Any local variable appearing for the first time on the left will be automatically defined. This is not true for tables and arrays which must be first created with verbs provided for that purpose. Note that mixed-mode arithmetic is accepted. Integers are expanded to real, or real truncated to integers, automatically, as required. Also note that division of one integer by a larger one always gives a zero result. However, if either numerator or denominator is real, real rather than integer division will be used. Thus

```
1/3 gives 0
1/3.0 gives .33333333
1.0/3 gives .33333333
```

The magnitude of an integer is limited to 32,767. A larger integer result is automatically converted to real or declared improper, depending on its intended disposition.

Expressions involving tables may use automatic indexing (also applicable to arrays) or name matching. This is done with the flags mentioned above. This notation really is a shorthand for DO-loops and should not be confused with matrix notation. Generalized matrix arithmetic can be readily programmed in DATAMAT but matrix operators are not provided in the language. (They are readily added by use of macros.) Suppose, for example, that G:A stands for a table with m rows and n columns and G:B for a table with n rows and p columns. Suppose G:C is an empty (i.e., all zero) table with m rows and p columns. Then the statement

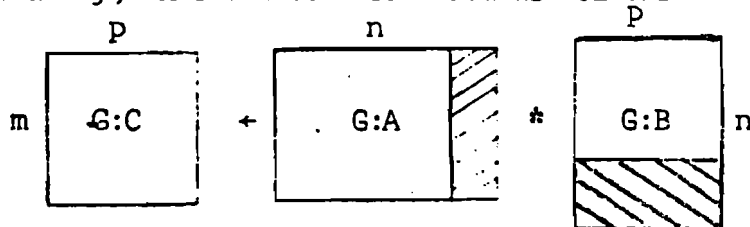
```
CALC  G:C(!1,!2)= G:A(!1,!2) *G:B(!1,!2)
```

multiplies individual elements of G:A and G:B and puts the results in G:C. The range of the automatic indices will be as follows:

$$!1 = 1, \dots, \min(m,n)$$

$$!2 = 1, \dots, \min(n,p)$$

Pictorially, this can be visualized as follows:



where the slanting lines represent unused elements. If one really wished to do matrix multiplication, the proper statement is:

```
CALC  G:C(!1,!2) = G:C(!1,!2) + G:A(!1,!3) * G:B(!3,!2)
```

Here,

$$!1 = 1, \dots, m$$

$$!2 = 1, \dots, p$$

$$!3 = 1, \dots, n$$

With name matching, the tables need not be conformable. This would be accomplished by writing "1, "2, "3 for !1, !2, !3 in the above. Of course, the tables could be used in other arrangements, for example, the factors could be multiplied row by row instead of row by column.

This generalization of matrix operators for tables is not as useful in practice as it is instructive. It discloses the implicit assumptions in standard matrix operations and displays, in the most abbreviated form possible, what is really involved. More virtuosity with the use of DATAMAT is achieved by thinking in terms of indexing and name-matching sets rather than in terms of matrix operations.

*AP*  
*EC*

## 2. Local Variables

Five types of local variables are used in DATAMAT. With one exception, these are automatically defined by their first appearance on the left side of a phrase. These types are as follows.

1. E-Variables. REAL\*8 (D-format)
2. I-Variables. INTEGER\*2 (H-format)
3. N-Variables. ALPHA\*8 (C-format)
4. Q-Variables. Value is T or F. A maximum of seven may be defined and their names are limited to seven characters.
5. O-Switches. Value is 0 or 1. There are 26 of these predefined as O:A, O:B,..., O:Z. Their initial value is 1. They may be used as either arithmetic or logic variables.

If a local variable (except O-switches) appears for the first on the left of a phrase and also on the right, the value taken on the right is as follows:

E-variables	0.0
I-variables	0
N-variables	8 blanks
Q-variables	F

The values of local variables cannot be saved from one DATAMAT session to another. When exit is made from DATAMAT, they vanish. This is also true of all other quantities created by DATAMAT except tables and models which may be enfiled.

Calculations are performed by the verbs CALC (for numerical values), MANIP (symbolic values), and LOGIC (logical values). However the distinction between CALC and MANIP, and to a lesser extent between these and LOGIC, is not sharply defined. For the most part, the value of an expression is converted to

the proper format for the result. The Q-variables are an exception and their use is not general. The O-switches may be used in both CALC and LOGIC statements. The set of symbolic manipulative functions is not valid in CALC but they are generally valid wherever else they would be useful.

If a result is a C-format quantity and the value of the expression is numeric, the integer part of the value is converted to 8-character EBCDIC code with no zero suppression. A value not less than  $10^8$  results in an error. A negative number is represented in 10s-complement form.

Thus 25.326 gives

C'00000025'

and -25 gives

C'99999975'

If the result is an O-switch, the binary units position of the value of the expression is used. Thus

25.326 gives 1

-25 gives 1

22 gives 0

-22 gives 0

### 3. Structuring Basic Data Tables

The use of DATAMAT centers around tables which contain numerical or symbolic values. However, it is the symbolic stubs and heads, with their implied indexing, which are the building blocks for a data management application.\*

---

\*See also Part I, Example 2.

15  
102

Suppose one is concerned with a model involving three plants, two raw material sources, six market regions and four time periods. One does not want to deal with basic tables each containing information about all 144 combinations. Rather, one wants some tables containing information about plants, some about materials, and others about markets. Time periods will be represented in a model mainly by replications with variations, such as level of demands and availability of materials.

Consider tables for representing plants. Although all plants may not be identical, a unified modelling scheme should underlie them all. Thus, although different plants may use some or all different materials, carry out different processes, and produce different market items, the scheme for representing these things should be universal. For example, a standard scheme is to use rows for representing input, processing and output streams, and columns for representing processes. For a complex plant even this may be too aggregated and one may want a table for each process, with columns, say, representing various raw material types, rows representing output streams, and the elements representing yields. One may even have to represent different modes of operation for the process, either with different tables or different sets of columns. Already we are beginning to imply a sizeable number of designators which it is clear must be coded in some fashion.

To keep a system with a large number of interrelated concepts operable, a certain discipline is demanded. In this connection certain special tables and rules of symbol formation can be extremely helpful.

Two such table types may be called dictionaries and catalogues.

They are based on the assumptions of master codes and position significance for symbol parts.\*

One begins by dividing all entities of interest into classes, for example: raw materials, products, time periods, production processes. Each entity class has an associated dictionary. For this purpose, H-tables are ideal. The stub consists of, say, 2-character codes for all possible entities in the class. The first column of the table has a 6- or 8-character mnemonic which is meaningful to a specialist. The next three or four columns are batched together to form 24- or 32-character text which is unambiguous to anyone in the field. The mnemonics can be used for reference purposes by analysts and the text used in preparing management or formal reports. An abbreviated example follows:

H:PLANT.DY	A1	B3
P1	NYEAST	LONG ISLAND PLANT
P2	NYWEST	BUFFALO PLANT
P3	GULF..	HOUSTON PLANT
P4	WCOAST	LOS ANGELES PLANT

\* See Palmer, K.H., An Integrated System for Handling Data Management, Matrix Generation, Solution Strategy and Report Writing for Large-Scale Linear Programming Problems in "Optimization Methods", R.W. Cottle and J. Krarup, eds., Univ. of London Press, 1974.



Note that the table name has been padded to 8 characters with a period and ends in DY. The convention here (only one of many possibilities) is that a table name with DY in positions 7 and 8 is a dictionary.

Note also that the stub names are only 2 characters, with no mnemonic significance. (You might prefer LI, BU, HO, LA instead.)

An important rule is that any data table have all stub names taken from one and only one dictionary stub, and likewise for head names. This leads to the concept of a catalogue table. One catalogue will be the catalogue of all data tables. Let us suppose that a catalogue's name is identified by CT in positions 7 and 8. A catalogue is an M-table with two columns which can have head names, say, STUB and HEAD. The stub of the catalogue is a list of all tables in the set. Suppose MT in positions 7 and 8 denotes an M-table of symbolic values and GT denotes a G-table of numeric values. Then the catalogue might appear as follows:

M:DATABSCT	STUB	HEAD
MARKT1GT PRICE.GT MARKT1MT ⋮	PRODCTDY PRODCTDY WHSE..DY ⋮	CUST1.DY REGIONDY CUST1.DY ⋮

(Ideally, the stubs should be alphabetized.) The meaning of a catalogue is as follows, using the first row of the example for specificity:

A G-TABLE exists called MARKT1GT (market 1 demands, say). All stub entries in this table are found in the stub of dictionary PRODCITY (products) and all head entries are found in the stub of dictionary CUST1.DY (customers).

One major purpose of such a catalogue is to be able to check whether changes to a table are legitimate. Such checking is readily programmed in DATAMAT statements. (Refer to the Palmer paper for further elaboration.)

Clearly, the concepts suggested above can be extended in many ways. It is extremely important that careful planning be done at the outset in laying out a system of tables. Attention must also be paid to how these will be used in matrix generation and report generation statements, as well as intermediate or ad hoc calculations.

#### 4. Forming, Maintaining and Using Tables

DATAMAT provides seven verbs explicitly for table formation and filing and three other verbs have options relating to table filing. A number of verbs may utilize table references, only one of which is specific to tables.

##### a. Table Construction Verbs

- (i) The simplest verb for constructing a table is called TABLE. The head is entered explicitly followed by each row beginning with its stub name. This method is very useful for small work tables (see Part I, Example 1) but tedious for larger ones. (See READTAB below.)
- (ii) A stub-only table can be constructed with the verb STUB. It can take the stub or head of another table, with symbolic functions (masking, filling, concatenating, etc.), to form the new stub. It will also extend or overlay the stub of an existing stub-only table.

10  
10  
10

- (iii) A null (empty) table can be constructed with the verb FORM. Either heads or stubs of existing tables can be combined with boolean set functions to form either the head or stub of the new table. Also, stylized heads and stubs can be created using a symbol concatenated with a running index.
- (iv) The verb READTAB reads a card-image file containing TABLE-verb statements and/or updating information. The file is more easily created with CMS EDIT facilities than with direct typing into DATAMAT. The updating facilities of READTAB are unique.
- (v) The verb REFORM creates a null table whose stub consists of LP identifiers from the currently-defined model. Either a mask or a bit map may be used for selection, or either all row or all column identifiers may be specified.

b. Table Filing and Deleting Verbs

- (i) The verb ENFILE (also used for models) has options for enfileing either one table, all tables, or all matching a mask. They may be listed by name and size at the same time. They are enfiled in the currently-defined TABLES file (see DEF below). There is no effect on tables in working storage. If an enfiled table replaces one on file, a note is typed.
- (ii) The verb RECALL recalls one table or all tables with a list option. Another option merely lists all tables on file without actually bringing them in to working storage. Both ENFILE and RECALL for one table have a renaming option. If a table of the same name (or alias) exists in working storage, it is first deleted.
- (iii) The verb ERASE erases one table in the file. It has no other function. (It must not be confused with DELETE.)
- (iv) The general verb DELETE has options for deleting one table or a list of them from working storage. This has no effect on any enfiled tables.
- (v) The general verb SET has an option for setting the file name for tables. The default is TABLES.

### c. Table Calculation Verbs

A large number of verbs utilize tables. However, for calculations on tables, as such, the following are used.

- (i) The verb DIMEN will return the integer index of any stub or head name or the number of rows or columns. If a name does not exist, the answer is 0.
- (ii) The general verb CALC allows G-table elements for both results and expressions. An elaborate variety of indexing is provided.
- (iii) The general verb MANIP allows M-table elements for results. Any table, including stubs and heads, may be used in the expression.
- (iv) The general verb LOGIC allows table references in expressions. This also applies to the IF verb. However, multiple indexing is meaningless even in a LOOP verb. (LOOP is a conditional DO-loop initiator.)
- (v) The verb CALC also moves values between a G-table and an array. A DATAMAT array is a FORTRAN-style array used with function and subroutines. Automatic indexing may be used but not name matching since arrays have no stubs or heads.

## 5. Model Generation and Revision

DATAMAT provides complete facilities for SESAME model generation, in other words, it may be used as what is usually termed a "matrix generator". In the SESAME environment, as with most large MP systems, one should properly make a distinction between an LP model and an LP matrix. A model is a complete representation stored in the user's data base; the matrix (there is only one at a time) is a particular specialization of a model residing on a work file for immediate use. In this sense, DATAMAT has nothing at all to do with the LP matrix.

DATAMAT deals with models in three distinct ways which must not be confused:

- (a) It can access any coefficient of an existing model for use in calculations. This is done with the referent prefixes A: and B: appearing in an expression. Here the existing model is treated as a source of data, just as a SESAME result file might be, and it has nothing in particular to do with model generation or revision.
- (b) DATAMAT can recall an entire existing model (REVISE verb) for revision. Once this has been done, the coefficients of the recalled model are not accessible directly as operands (though the original enfiled version is). There are three variants of REVISE:
  - (i) The argument DUMMY with REVISE causes only the list of INDIRECT names to be recalled and formed into a table. This is useful for revising the indirect vector used by SESAME procedures.
  - (ii) The verb SUBMODEL recalls only the row definitions of a model and changes their types in preparation for creating a decomposition submodel.
  - (iii) The verb MERGE recalls and merges an entire model with one previously recalled or initiated.
- (c) DATAMAT can create new model components or change existing ones. This is done with the model generation and revision verbs. Before these verbs are legal to use, one of the verbs REVISE, SUBMODEL or NEWMODEL must have been executed. The latter is used when one is starting a new model from scratch.

The model in working storage - revised or created as described above - must be enfiled (ENFILE verb) if it is to be retained. Since there is no other purpose for having a model in working storage, it must always be enfiled unless some gross error has been made in its construction and it is desired to start over. Although the general verb DELETE has an option for deleting the model in working storage, this is unnecessary.

The model in working storage is automatically deleted by any of the following:

ENFILE (of model)  
 REVISE, SUBMODEL or NEWMODEL  
 EXIT (from DATAMAT)

Although a series of MERGE statements may be used after REVISE or NEWMODEL, note that the sequence


```

REVISE or NEWMODEL
MERGE
ENFILE
MERGE
ENFILE:
:

```

is illegal. Enfiling a model is a destructive process to the model in working storage. This model is similar to but not identical with an enfiled model in structure. In particular, the enfiling process re-sorts the working model into the same order which is created by the SESAME procedure CONVERT and is essential to other procedures, particularly SETUP.

The verbs mentioned above deal with two different model designations on separately designated files. REVISE, SUBMODEL and MERGE always access the file designated as DDOLDMD and REVISE and SUBMODEL recall the model designated as OLDMD. (MERGE requires an argument naming the model to be merged.) The verb ENFILE; on the other hand, writes to the file designated DDMDL (except when the argument SUBMODEL is used in which case it writes to the DDOLDMD file and also modifies the OLDMD model). ENFILE



requires an argument to name the new model. Of course, this can be the same as the OLDMOD designation, if desired, and the DDOLDMD and DDMODEL files may also be the same. These designations are controlled by options with the general verb SET. (See write-up of SET verb for correspondence with SESAME CR cells. However, it is always safer to use the SET verb when in DATAMAT rather than to rely on prior CR settings in SESAME.)

Many of the model generation verbs are simply LP component names, viz:

ROW	defines one or more LP row identifiers and types and may define one or more columns and also the coefficients in the implied substructure.
COL	defines one or more LP column identifiers and types and may define one or more rows and also the coefficients in the implied substructure.
RHS	defines one or more LP RHS identifiers and may define one or more rows and also the elements in the implied RHS columns.
GUB	defines one GUB set header (identifier) and type, and optionally the GUB value.
PRSET	defines one pricing set header (identifier).
MARKER	defines one marker column (identifier).
RANGE	defines one or more range sets and their elements, and may define new row identifiers.
BOUND	defines one or more bound sets and the bound values for each set. The referenced columns must be defined elsewhere.

Wherever a new definition may be made in the above, an old one is first checked for. If an old definition exists and the statement makes a nontrivial change in any type or value, a note is output. Changing a free row or a standard column to some other type is regarded as non-exceptional. If new coefficients, elements or values are created (where no old one existed), no note is output.

The general verb DELETE has options for all the above component types and deletes the named component from the model in working storage. DELETE always applies to entire components; individual elements may be deleted by redefining them as zero. In the case of ranges and bounds, it is impossible to delete individual values since a zero value has a definite meaning. However, a range value for a deleted row or a bound value for a deleted column will be excised when ENFILE is executed.

Three additional verbs deal with sets of vectors and are primarily for revision of an old model:

INSERT This verb defines the point at which new rows, columns or RHS-s are to be inserted in the final sequencing.\*

ENDINS terminates all existing INSERTS.\*

COMBINE This verb creates a linear combination of two rows, two columns or two RHS-s in the form  $V_1 + V_2 * p$  where  $V_1$  and  $V_2$  are vectors and  $p$  is a scalar.

Its principle use is to combine base and change vectors from the parametric algorithms.

The result vector can be new or the same as either  $V_1$  or  $V_2$  which may also be the same. Thus COMBINE can also be used for scaling. For example,

$$V_1 = V_1 + V_1 * .5 \text{ multiplies } V_1 \text{ by } 1.5.$$

---

\*The verbs INSERT and ENDINS should be used sparingly as they can greatly increase processing time. It is almost never necessary to control order except for GUB and FPSET sets.



See the Palmer reference (section 3 above) for useful suggestions on structuring LP identifiers. The same kind of planning used for table stubs and heads should be carried over into LP nomenclature.

The final verb in this set is quite different from the others since it deals with a special category of model components, namely indirect values. There is a rather intricate interplay among model construction, model revision, matrix setup and matrix modification with respect to indirect values. The DATAMAT verb INDIRECT facilitates working with them. Although it partly duplicates the functions of the SESAME procedure VALUES, it is indispensable for models created by DATAMAT.

Indirect names must be defined before other parts of a model are created. In the SESAME procedure CONVERT, this is ensured by requiring the INDIRECT section to be first in the input file. Such an absolute requirement is not appropriate with the greater flexibility of DATAMAT but nevertheless a similar requirement must be enforced. There is the further difficulty of specifying the indirect names: to simply require one to type them in as a list would insulate them from other useful facilities.

The situation is handled as follows. The list of indirect names is specified as the head of a G-table and the values, if defined, are in some row of this table (which may have more rows or not). If one is creating a new model or installing indirect names for the first time in an old model, the verb INDIRECT with option PUT is executed, naming the

table defining the indirects. This must be done before any generation verbs referring to indirects are executed. Any associated values are not recorded in the model but may be put into the indirect vector in either of two ways:

- (1) If the SESAME procedure VALUES has been previously executed, the verb CALC with a special result operand convention may be used. The same operand convention for the expression with CALC may be used to retrieve values from the indirect vector, say to put into the appropriate row of the G-table.
- (2) The SESAME procedure VALUES may subsequently be used in standard fashion after the model has been enfiled by DATAMAT.

When REVISE or SUBMODEL is executed and the recalled model has an indirect branch, a G-table with one row is automatically created with an internally generated name. (The situation is more complicated with MERGE. See MERGE write-up.) The row is set to zeroes. Execution of the verb INDIRECT with option GET and specifying a G-table name, merely causes the generated table to be renamed. (If another table by that name exists, it is deleted.)

One may subsequently execute INDIRECT with option PUT naming another table. The indirect indexes will be translated and any old names not appearing in the new table head will be deleted. (This happens when ENFILE is executed.) Any remaining elements referring to these names (via the old index) will be treated as zero.

INDIRECT GET and INDIRECT PUT may each be executed at most once and, if both, GET must be first.

NE  
EC

## 6. Calculations with Model and Result File Quantities

It has already been mentioned that elements of an enfiled model (the one currently designated MODEL) may be referenced with the prefixes A: (for structural coefficients) and B: (for RHS elements). Twelve other prefixes are used for references to a result file. (Another prefix is used to refer to a SESAME general map. Thus 15 of the 26 prefixes refer to enfiled quantities.)

The reason so many prefixes are used for result files is that these files contain such varied information. The file referenced is the one currently designated DDRESLT and the main branch used is for the model currently designated MODEL. Below this, however, there are still several paths. The prefix automatically determines whether the reference is to a solution or to a tableau. There are two ways to specify the case name: either with an option of the verb SET, or by specifying case directly in the referent. The latter overrides the former without nullifying it. The subsets of the prefixes are as follows:

### (a) Solution ROW section.

P:	Dual activity ( $\pi$ )
U:	Logical activity (slack or surplus)
V:	Row sum ( $\sum a_{ij}X_j$ )
W:	Row lower limit, upper limit, or status

## (b) Solution COLUMN section

C: Input cost (possibly composite)  
 D: Reduced cost, or dual slack activity ( $d_j$ )  
 R: GUB set value (implied RHS for GUB row)  
 X: Structural activity (LP X-variable)  
 Y: Lower limit, upper limit, or status for X-variable

## (c) Solution CR section

Z: One of a subset of CR values recorded  
 with a solution case.

## (d) Tableau Branch

S: Same as Z: but for a tableau case.  
 T: Tableau value, designated by stub, head.

These referents may occur in practically any expression where they make sense.

There is one verb (DOT) and a summation convention for CALC for calculations especially appropriate to enfiled models and results.

These are as follows.

- a. The DOT verb computes inner products for the following combinations of prefixes (DOT expressions are special; see DOT writeup):

P: by A:  $\sum_i \pi_i a_{ij}$  for fixed  $j$ .

P: by B:  $\sum_i \pi_i b_{ik}$  for fixed  $k$ .

A: by X:  $\sum_j a_{ij} X_j$  for fixed  $i$

C: by X:  $\sum_j c_j X_j$  ( $c_j$  = composite OBJ row)

Cases are specified for P:, X: and C:. The two cases in the



last form need not be the same. The range of summation may be restricted with either a mask or a bit map (J: referent). The result may become the value of an E: variable or a G: table element. Multiple indexing may not be used directly but may be effected by use of LOOP.

2. A mask or a bit map may be used with any of the following referent prefixes in a CALC expression to denote summation:

A:, B:	Sum over all model coefficients whose identifiers match the mask(s) or have their bit on in the map. Double summation is possible.
P:, U:, V:, W: C:, D:, R:, X:, Y:	Sum over all results values whose identifiers match the mask or have their bit on in the map. Only single summation is possible. (For W: and Y: only the LL or UL options are meaningful.)
T:	Sum over all tableau values whose stub or head names match the mask(s). Double summation is possible. A bit map may <u>not</u> be used since tableau stubs and heads are not related to bit maps.

Note: Two maps may not be used simultaneously. If this is attempted, the second one named (from left to right) will be used and no error is flagged. The same map may be used for both LP rows and LP columns with prefix A:. For prefix B:, a bit map is legal only for LP rows, not LP RHS-s.